Temporal Parallelization of Bayesian Smoothers

Simo Särkkä, *Senior Member, IEEE,* Ángel F. García-Fernández

***Abstract*— This paper presents algorithms for temporal parallelization of Bayesian smoothers. We define the elements and the operators to pose these problems as the solutions to all-prefix-sums operations for which efficient parallel scan-algorithms are available. We present the temporal parallelization of the general Bayesian filtering and smoothing equations and specialize them to linear/Gaussian models. The advantage of the proposed algorithms is that they reduce the linear complexity of standard smoothing algorithms with respect to time to logarithmic.**

***Index Terms*— Bayesian smoothing, Kalman filtering and smoothing, parallel computing, parallel scan, prefix sums**

I. Introduction

Parallel computing is rapidly transforming from a scientists' computational tool to a general purpose computational paradigm. The availability of affordable massively-parallel graphics processing units (GPUs) as well as widely-available parallel grid and cloud computing systems [1]–[3] drive this transformation by bringing parallel computing technology to everyday use. This creates a demand for parallel algorithms that can harness the full power of the parallel computing hardware.

Stochastic state-space models allow for modeling of time-behaviour and uncertainties of dynamic systems, and they have long been used in various tracking, automation, communications, and imaging applications [4]–[8]. More recently, they have also been used as representations of prior information in machine learning setting (see, e.g., [9]). In all of these applications, the main problem can be mathematically formulated as a state-estimation problem on the stochastic model, where we estimate the unknown phenomenon from a set of noisy measurement data. Given the mathematical problem, the remaining task is to design efficient computational methods for solving the inference problems on large data sets such that they utilize the computational hardware as effectively as possible.

Bayesian filtering and smoothing methods [6] provide the classical [10] solutions to state-estimation problems which are computationally optimal in the sense that their computational complexities are linear with respect to the number of data points. Although these solutions are optimal for single central processing unit (CPU) systems, due to the sequential nature of the algorithms, their complexity remains linear also in parallel multi-CPU systems. However, genuine parallel algorithms are often capable to perform operations in logarithmic number of steps by massive parallelization of the operations. More precisely, their span-complexity [3], that is, the number of computational steps as measured by a wall-clock, is often logarithmic with respect to the number of data points. However, their total number of operations, the work-complexity, is still linear as all the data points need to be processed.

Despite the long history of state-estimation methods, the existing parallelization methods have concentrated on parallelizing the subproblems arising in Bayesian filtering and smoothing methods, but there is a lack of algorithms that are specifically designed for solving state-estimation problems in parallel architectures. There are, however, some existing approaches for parallelizing Kalman type of filters as well as particle filters. One approach was studied in [11] and [12] is to parallelize the corresponding batch formulation, which leads to sub-linear computational methods, because the matrix computations can be parallelized. If the state-space of the Kalman filter is large, it is then possible to speed up the matrix computations via parallelization [13], [14]. Particle filters can also be parallelized over the particles [15], [16] the bottleneck being the resampling step. In some specific cases such as in multiple target tracking [17] it is possible to develop parallelized algorithms by using the structure of the specific problem.

The contribution of this article is to propose a novel general algorithmic framework for parallel computation of Bayesian smoothing solutions for state space models. We also present algorithms for parallelizing the Bayesian filtering solutions, but our focus is in smoothing, because the parallel computation is done off-line in the sense that all the data needs to be available during the parallel computations and it cannot arrive sequentially. Our approach to parallelization differs from the aforementioned approaches in the aspect that we replace the whole Bayesian filtering and smoothing formalism with another, parallelizable formalism. We replace the Bayesian filtering and smoothing equations [4], [6] with another set of equations that can be combined with so-called scan or prefix-sums algorithm [2], [18]–[20], which is one of the fundamental algorithm frameworks in parallel computing. The advantage of this is that it allows for reduction of the linear $O(n)$ complexity of batch filtering and smoothing algorithms to logarithmic $O(\log n)$ span-complexity in the number n of data points. Based on the novel formulation we develop parallel algorithms for computing the filtering and smoothing solutions to linear Gaussian systems with the logarithmic span-complexity. As this parallelization is done in temporal direction, the individual steps of the resulting algorithm could further be parallelized in the same way as Kalman filters and particle filters have previously been parallelized [13]–[17].

The organization of the article is the following. In Section II we review the classical Bayesian filtering and smoothing methodology as well as the parallel scan algorithm for computing prefix sums. In Section III we present the general framework for parallelizing Bayesian filtering and smoothing methods. Section IV is concerned with specializing the general framework to linear Gaussian systems. Numerical example of linear/Gaussian systems is given in Section V and finally Section VI concludes the article along with discussion on various aspects of the methodology.

II. Background

A. Bayesian filtering and smoothing

Bayesian filtering and smoothing methods [6] are algorithms for statistical inference in probabilistic state-space models of the form

$$\begin{aligned} x_k &\sim p(x_k \mid x_{k-1}), \\ y_k &\sim p(y_k \mid x_k). \end{aligned} \tag{1}$$

with $x_0 \sim p(x_0)$. Above, the state $x_k \in \mathbb{R}^{n_x}$ at time step k evolves as a Markov process with transition density $p(x_k \mid x_{k-1})$. State x_k is observed by the measurement $y_k \in \mathbb{R}^{n_y}$ whose density is $p(y_k \mid x_k)$.

S. Särkkä is with the Department of Electrical Engineering and Automation, Aalto University, 02150 Espoo, Finland (email: simo.sarkka@aalto.fi).

Ángel F. García-Fernández is with the Department of Electrical Engineering and Electronics, University of Liverpool, Liverpool L69 3GJ, United Kingdom (email: angel.garcia-fernandez@liverpool.ac.uk).

The objective of Bayesian filtering is to compute the posterior density $p(x_k \mid y_{1:k})$ of the state x_k given the measurements $y_{1:k} = (y_1, \ldots, y_k)$ up to time step k. Given the measurements up to a time step n, the objective is smoothing is to compute the density $p(x_k \mid y_{1:n})$ of the state x_k for $k < n$.

The key insight of Bayesian filters and smoothers is that the computation of the required densities can be done in linear $O(n)$ number of computational steps by using recursive (forward) filtering and (backward) smoothing algorithms. This is significant, because a naive computation of the posterior distribution would typically take at least $O(n^3)$ computational steps.

The Bayesian filter is a sequential algorithm, which iterates the following prediction and update steps:

$$p(x_k \mid y_{1:k-1}) = \int p(x_k \mid x_{k-1})\, p(x_{k-1} \mid y_{1:k-1})\, \mathrm{d}x_{k-1}, \quad (2)$$

$$p(x_k \mid y_{1:k}) = \frac{p(y_k \mid x_k)\, p(x_k \mid y_{1:k-1})}{\int p(y_k \mid x_k)\, p(x_k \mid y_{1:k-1})\, \mathrm{d}x_k}. \quad (3)$$

Given the filtering outputs for $k = 1, \ldots, n$, the Bayesian forward-backward smoother consists of the following backward iteration for $k = n-1, \ldots, 1$:

$$p(x_k \mid y_{1:n}) = p(x_k \mid y_{1:k}) \int \frac{p(x_{k+1} \mid x_k)\, p(x_{k+1} \mid y_{1:n})}{p(x_{k+1} \mid y_{1:k})} \mathrm{d}x_{k+1}. \quad (4)$$

When applied to a batch of data of size n, the computational complexity of the filter and smoother is $O(n)$ as they perform n sequential steps in forward and backward directions when looping over the data. The Kalman filter and Rauch–Tung–Striebel (RTS) smoother [21], [22] are the solutions to these recursions when the transition densities are linear and Gaussian. The filtering and smoothing equations can also be analogously solved in closed form for discrete-state models [7]. In this paper, we show how to parallelize the previous recursions using parallel scan-algorithm, which is reviewed next.

B. Parallel scan-algorithm

The parallel-scan algorithm [20] is a general parallel computing framework that can be used to convert sequential $O(n)$ algorithms with certain associative property to $O(\log n)$ parallel algorithms. The algorithm was originally developed for computing prefix sums [18], where it uses the associative property of summation. The algorithm has since been generalized to arbitrary associative binary operators and it is used as the basis of multitude of parallel algorithms including sorting, linear programming, and graph algorithms. This kind of algorithms are especially useful in GPU-based computing systems and they are likely to be fundamental algorithms in a many future parallel computing systems.

The problem that the parallel-scan algorithm [20] solves is the all-prefix-sums operation, which is defined next.

Definition 1: Given a sequence of elements $(a_1, a_2, \ldots, a_n)$, where a_i belongs to a certain set, along with an associative binary operator $\otimes$ on this set, the all-prefix-sums operation computes the sequence

$$(a_1, a_1 \otimes a_2, \ldots, a_1 \otimes \cdots \otimes a_n). \quad (5)$$

For example, if we have $n = 4$, $a_i = i$, and $\otimes$ denotes the ordinary summation, the all-prefix-sums are $(1, 3, 6, 10)$. If $\otimes$ denotes the subtraction, the all-prefix-sums are $(1, -1, -4, -8)$. It should be noted that the operator is not necessarily commutative so we use a product symbol, as matrix products are not commutative, instead of a summation symbol.

The all prefix-sums operation can be computed sequentially by processing one element after the other. However, this direct sequential iteration inherently takes $O(n)$ time. We can now see the analogy of the iteration to the Bayesian filter discussed in previous section – both of the algorithms have linear $O(n)$ complexity, because they need to loop over all the elements in forward direction. A similar argument applies to the Bayesian smoothing pass.

```
// Save the input:
for i ← 1 to n do                                   ▷ Compute in parallel
    b_i ← a_i
end for
// Up sweep:
for d ← 0 to log_2 n − 1 do
    for i ← 0 to n − 1 by 2^{d+1} do                ▷ Compute in parallel
        j ← i + 2^d
        k ← i + 2^{d+1}
        a_k ← a_j ⊗ a_k
    end for
end for
// Down sweep:
for d ← log_2 n − 1 to 0 do
    for i ← 0 to n − 1 by 2^{d+1} do                ▷ Compute in parallel
        j ← i + 2^d
        k ← i + 2^{d+1}
        t ← a_j
        a_j ← a_k
        a_k ← a_k ⊗ t
    end for
end for
// Final pass:
for i ← 1 to n do                                   ▷ Compute in parallel
    a_i ← a_i ⊗ b_i
end for
```

Fig. 1. Parallel scan algorithm for in-place transformation of the sequence $(\boldsymbol{a_i})$ into its all-prefix-sums in $\boldsymbol{O(\log n)}$ span-complexity. Note that the algorithm in this forms assumes that $\boldsymbol{n}$ is a power of $\mathbf{2}$, but it can easily be generalized to an arbitrary $\boldsymbol{n}$.

Fortunately, the all-prefix-sum sums operation can be computed in parallel in $O(\log n)$ span-time by using *up-sweep* and *down-sweep* algorithms [20] shown in Fig. 1. These algorithms correspond to up and down traversals in a binary tree which are used for computing partial (generalized) sums of the elements. A final pass is then used to construct the final result. The algorithms can be used for computing all-prefix-sums (5) for an arbitrary associative operator $\otimes$.

III. Parallel Bayesian filtering and smoothing

In this section, we explain how to define the elements and the binary operators to be able to perform Bayesian filtering and smoothing using parallel scan algorithms.

A. Bayesian filtering

In order to perform parallel Bayesian filtering, we need to find the suitable element a_k and the binary associative operator $\otimes$. As we will see in this section, an element a consists of a pair $(f, g) \in \mathcal{F}$ where $\mathcal{F}$ is

$$\mathcal{F} = \left\{ (f, g) : \int f(y \mid z)\, \mathrm{d}y = 1 \right\}, \quad (6)$$

and $f : \mathbb{R}^{n_x} \times \mathbb{R}^{n_x} \to [0, \infty)$ represents a conditional density, and $g : \mathbb{R}^{n_x} \to [0, \infty)$ represents a likelihood.

Definition 2: Given two elements $(f_i, g_i) \in \mathcal{F}$ and $(f_j, g_j) \in \mathcal{F}$, the binary associative operator $\otimes$ for Bayesian filtering is

$$(f_i, g_i) \otimes (f_j, g_j) = (f_{ij}, g_{ij}),$$

where

$$f_{ij}(x \mid z) = \frac{\int g_j(y) f_j(x \mid y) f_i(y \mid z) \,\mathrm{d}y}{\int g_j(y) f_i(y \mid z) \,\mathrm{d}y},$$
$$g_{ij}(z) = g_i(z) \int g_j(y) f_i(y \mid z) \,\mathrm{d}y.$$

The proof that $\otimes$ has the associative property is given in Appendix I.

Theorem 3: Given the element $a_k = (f_k, g_k) \in \mathcal{F}$ where

$$f_k(x_k \mid x_{k-1}) = p(x_k \mid y_k, x_{k-1}),$$
$$g_k(x_{k-1}) = p(y_k \mid x_{k-1}),$$

$p(x_1 \mid y_1, x_0) = p(x_1 \mid y_1)$, and $p(y_1 \mid x_0) = p(y_1)$, the k-th prefix sum is

$$a_1 \otimes a_2 \otimes \cdots \otimes a_k = \begin{pmatrix} p(x_k \mid y_{1:k}) \\ p(y_{1:k}) \end{pmatrix}.$$

Theorem 3 is proved in Appendix I. Theorem 3 implies that we can parallelise the computation of all filtering distributions $p(x_k \mid y_{1:k})$ and the marginal likelihoods $p(y_{1:k})$, of which the latter ones can be used for parameter estimation [6].

Remark 4: If we only know $p(y_k \mid x_{k-1})$ up to a proportionality constant, which means that $g_k(x_{k-1}) \propto p(y_k \mid x_{k-1})$, we can still recover the filtering density $p(x_k \mid y_{1:k})$ by the above operations. However, we will not be able to recover the marginal likelihoods $p(y_{1:k})$. We can nevertheless recover $p(y_{1:k})$ by an additional parallel pass, as will be explained in Section III-C.

B. Bayesian smoothing

The Bayesian smoothing pass requires that the filtering densities have been obtained beforehand. In smoothing, we consider a different type of element a and binary operator $\otimes$ than those used in filtering. As we will see in this section, an element a is a function $a : \mathbb{R}^{n_x} \times \mathbb{R}^{n_x} \to [0, \infty)$ that belongs to the set

$$\mathcal{S} = \left\{ a : \int a(x \mid z) \,\mathrm{d}x = 1 \right\}.$$

Definition 5: Given two elements $a_i \in \mathcal{S}$ and $a_j \in \mathcal{S}$, the binary associative operator $\otimes$ for Bayesian smoothing is

$$a_i \otimes a_j = a_{ij},$$

where

$$a_{ij}(x \mid z) = \int a_i(x \mid y) a_j(y \mid z) \,\mathrm{d}y.$$

The proof that $\otimes$ has the associative property is included in Appendix II.

Theorem 6: Given the element $a_k = p(x_k \mid y_{1:k}, x_{k+1}) \in \mathcal{S}$, with $a_n = p(x_n \mid y_{1:n})$, we have that

$$a_k \otimes a_{k+1} \otimes \cdots \otimes a_n = p(x_k \mid y_{1:n}).$$

Theorem 6 is proved in Appendix II. Theorem 6 implies that we can compute all smoothing distributions in parallel form. However, it should be noted we should apply the parallel scan algorithm with elements in reverse other, that is, with elements $b_k = a_{n-k+1}$, so that the prefix-sums $b_1 \otimes \cdots \otimes b_k$ recover the smoothing densities.

C. Additional aspects

We proceed to discuss additional aspects of the previous formulation of filtering and smoothing. In Section III-A, it was indicated that the marginal likelihood $p(y_{1:n})$ is directly available from the parallel scan algorithm if $g_k(x_{k-1}) = p(y_k \mid x_{k-1})$. However, sometimes we only know $p(y_k \mid x_{k-1})$ up to a proportionality constant so $g_k(x_{k-1}) \propto p(y_k \mid x_{k-1})$, as will happen in Section IV. Although in this case, the parallel scan Bayesian filtering algorithm provides us with the filtering densities but not the marginal likelihood $p(y_{1:n})$, we can still recover the marginal likelihoods as follows. We first run the parallel filtering algorithm to recover all filtering distributions $p(x_k \mid y_{1:k})$ for $k = 1$ to n and then, we perform the following decomposition for $p(y_{1:n})$

$$p(y_{1:n}) = \prod_{k=1}^{n} p(y_k \mid y_{1:k-1}),$$

where

$$p(y_k \mid y_{1:k-1}) = \int p(y_k \mid x_k) p(x_k \mid y_{1:k-1}) \,\mathrm{d}x_k.$$

Each factor $p(y_k \mid y_{1:k-1})$ can be computed in parallel using the predictive density $p(x_k \mid y_{1:k-1})$ and the likelihood $p(y_k \mid x_k)$. We can then recover all $p(y_{1:k})$ by $O(\log n)$ parallel recursive pairwise multiplications of the adjacent terms.

It is also possible to perform the parallelization at block level instead of at individual element level. When using the parallel scan algorithm, we do not need to assign each single-measurement element to a single computational node, but instead we can perform initial computations in blocks such that a single node processes a block of measurements before combining the results with other blocks. The results of the blocks can then be used as the elements in the parallel-scan algorithm. This kind of procedure corresponds to selecting the elements for the scan algorithm to consist of blocks of length l:

$$a_k = \begin{pmatrix} p\left(x_{lk} \mid y_{l(k-1)+1:kl}, x_{l(k-1)}\right) \\ p\left(y_{l(k-1)+1:kl} \mid x_{l(k-1)}\right) \end{pmatrix}$$

in filtering and

$$a_k = p\left(x_{lk} \mid y_{1:l(k+1)-1}, x_{l(k+1)}\right) \tag{7}$$

in smoothing instead of the corresponding terms with $l = 1$. A practical advantage of this is that we can more easily distribute the computations to a limited number of computational nodes while still getting the optimal speedup from parallelization.

IV. Parallel linear/Gaussian filter and smoother

The parallel linear/Gaussian filter and smoother are obtained by particularising the element a and binary operator $\otimes$ for Bayesian filtering and smoothing explained in the previous section to linear/Gaussian systems. The sequential versions of these algorithms correspond to the Kalman filter and the RTS smoother.

We consider the linear/Gaussian state space model

$$x_k = F_{k-1} x_{k-1} + u_{k-1} + q_{k-1},$$
$$y_k = H_k x_k + d_k + r_k,$$

where $F_{k-1} \in \mathbb{R}^{n_x \times n_x}$ and $H_k \in \mathbb{R}^{n_y \times n_x}$ are known matrices, $u_{k-1} \in \mathbb{R}^{n_x}$ and $d_k \in \mathbb{R}^{n_y}$ are known vectors, and q_{k-1} and r_k are zero-mean, independent Gaussian noises with covariance matrices $Q_{k-1} \in \mathbb{R}^{n_x \times n_x}$ and $R_k \in \mathbb{R}^{n_y \times n_y}$. The initial distribution is given as $x_0 \sim \mathrm{N}(m_0, P_0)$. With this model, we have that

$$p(x_k \mid x_{k-1}) = \mathrm{N}(x_k; F_{k-1} x_{k-1} + u_{k-1}, Q_{k-1}), \tag{8}$$
$$p(y_k \mid x_k) = \mathrm{N}(y_k; H_k x_k + d_k, R_k). \tag{9}$$

In this section, we use the notation $\mathrm{N}_I(\cdot; \eta, J)$ to denote a Gaussian density parameterised in information form so that η is the information vector and J is the information matrix. If a Gaussian distribution has mean $\overline{x}$ and covariance matrix P, its parameterisation in information form is $\eta = P^{-1}\overline{x}$ and $J = P^{-1}$. This parametrization corresponds to so-called information form of Kalman filter [23]. We also use I_{n_x} to denote an identity matrix of size n_x.

A. Linear/Gaussian filtering

We first describe the representation of an element $a_k \in \mathcal{F}$ for filtering in linear and Gaussian systems by the following lemma.

Lemma 7: For linear/Gaussian systems, the element $a_k \in \mathcal{F}$ for filtering becomes

$$f_k(x_k \mid x_{k-1}) = p(x_k \mid y_k, x_{k-1}) = \mathrm{N}(x_k; A_k x_{k-1} + b_k, C_k),$$
$$g_k(x_{k-1}) = p(y_k \mid x_{k-1}) \propto \mathrm{N}_I(x_{k-1}; \eta_k, J_k),$$

where the parameters of the first term are given for $k > 1$ as

$$\begin{aligned} A_k &= (I_{n_x} - K_k H_k) F_{k-1}, \\ b_k &= u_{k-1} + K_k (y_k - H_k u_{k-1} - d_k), \\ C_k &= (I_{n_x} - K_k H_k) Q_{k-1}, \\ K_k &= Q_{k-1} H_k^\top S_k^{-1}, \\ S_k &= H_k Q_{k-1} H_k^\top + R_k, \end{aligned} \tag{10}$$

and for $k = 1$ as

$$\begin{aligned} m_1^- &= F_0 m_0 + u_0, \\ P_1^- &= F_0 P_0 F_0^\top + Q_0, \\ S_1 &= H_1 P_1^- H_1^\top + R_1, \\ K_1 &= P_1^- H_1^\top S_1^{-1}, \\ A_1 &= 0, \\ b_1 &= m_1^- + K_1[y_1 - H_1 m_1^- - d_1], \\ C_1 &= P_1^- - K_1 S_1 K_1^\top. \end{aligned} \tag{11}$$

The parameters of the second term are given as

$$\begin{aligned} \eta_k &= F_{k-1}^\top H_k^\top S_k^{-1} (y_k - H_k u_{k-1} - d_k), \\ J_k &= F_{k-1}^\top H_k^\top S_k^{-1} H_k F_{k-1}, \end{aligned} \tag{12}$$

for $k = 1, \ldots, n$.

In Lemma 7, densities $p(x_k \mid y_k, x_{k-1})$ and $p(y_k \mid x_{k-1})$ are obtained by applying the Kalman filter update with measurement y_k, distributed according to (9), applied to the density $p(x_k \mid x_{k-1})$ in (8) and matching the terms. For the first step we have applied the Kalman filter prediction and update steps starting from $x_0 \sim \mathcal{N}(m_0, P_0)$ and matched the terms.

Therefore, an element a_k can be parameterised by $(A_k, b_k, C_k, \eta_k, J_k)$, which can be computed for each element in parallel. Also, it is relevant to notice that if the system parameters $(F_k, u_k, Q_k, H_k, d_k, R_k)$ do not depend on the time step k, the only parameters of a_k that depend on k are b_k and η_k, as they depend on the measurement y_k.

Lemma 8: Given two elements $(f_i, g_i) \in \mathcal{F}$ and $(f_j, g_j) \in \mathcal{F}$, with parameterisations

$$\begin{aligned} f_i(y \mid z) &= \mathrm{N}(y; A_i z + b_i, C_i), \\ g_i(z) &\propto \mathrm{N}_I(z; \eta_i, J_i), \\ f_j(y \mid z) &= \mathrm{N}(y; A_j z + b_j, C_j), \\ g_j(z) &\propto \mathrm{N}_I(z; \eta_j, J_j), \end{aligned}$$

the binary operator $\otimes$ for filtering becomes

$$(f_i, g_i) \otimes (f_j, g_j) = (f_{ij}, g_{ij}),$$

where

$$f_{ij}(x \mid z) = \mathrm{N}(x; A_{ij} z + b_{ij}, C_{ij}), \tag{13}$$
$$g_{ij}(z) \propto \mathrm{N}_I(z; \eta_{ij}, J_{ij}), \tag{14}$$

with

$$\begin{aligned} A_{ij} &= A_j (I_{n_x} + C_i J_j)^{-1} A_i, \\ b_{ij} &= A_j (I_{n_x} + C_i J_j)^{-1} (b_i + C_i \eta_j) + b_j, \\ C_{ij} &= A_j (I_{n_x} + C_i J_j)^{-1} C_i A_j^\top + C_j, \\ \eta_{ij} &= A_i^\top (I_{n_x} + J_j C_i)^{-1} (\eta_j - J_j b_i) + \eta_i, \\ J_{ij} &= A_i^\top (I_{n_x} + J_j C_i)^{-1} J_j A_i + J_i. \end{aligned}$$

The proof is provided in Appendix III.

B. Linear/Gaussian smoothing

We first describe the representation of an element $a_k \in \mathcal{S}$ for smoothing in linear and Gaussian systems by the following lemma.

Lemma 9: For linear/Gaussian systems, the element $a_k \in \mathcal{S}$ for smoothing becomes

$$\begin{aligned} a_k(x_k \mid x_{k+1}) &= p(x_k \mid y_{1:k}, x_{k+1}) \\ &= \mathrm{N}(x_k; E_k x_{k+1} + g_k, L_k), \end{aligned}$$

where for $k < n$

$$\begin{aligned} E_k &= P_k F_k^\top \left(F_k P_k F_k^\top + Q_k\right)^{-1}, \\ g_k &= \overline{x}_k - E_k (F_k \overline{x}_k + u_k), \\ L_k &= P_k - E_k F_k P_k, \end{aligned}$$

and for $k = n$ we have

$$\begin{aligned} E_n &= 0, \\ g_n &= \overline{x}_n, \\ L_n &= P_n. \end{aligned}$$

Above, $\overline{x}_k$ and P_k are the filtering mean and covariance matrix at time step k, such that $p(x_k \mid y_{1:k}) = \mathrm{N}(x_k; \overline{x}_k, P_k)$.

Lemma 9 is obtained by performing a Kalman filter update on density $p(x_k \mid y_{1:k})$ with an observation x_{k+1}, whose distribution is given by (8). Element a_k for smoothing with linear/Gaussian systems can be parameterised as $a_k = (E_k, g_k, L_k)$.

Lemma 10: Given two elements $a_i \in \mathcal{S}$ and $a_j \in \mathcal{S}$ with parameterisation

$$a_i(y \mid z) = \mathrm{N}(y; E_i z + g_i, L_i),$$

the binary operator $\otimes$ for smoothing becomes

$$a_i \otimes a_j = a_{ij},$$

where

$$\begin{aligned} a_{ij}(x \mid z) &= \int a_i(x \mid y)\, a_j(y \mid z)\, \mathrm{d}y \\ &= \int \mathrm{N}(x; E_i y + g_i, L_i)\, \mathrm{N}(y; E_j z + g_j, L_j)\, \mathrm{d}y \\ &= \mathrm{N}(x; E_{ij} z + g_{ij}, L_{ij}), \end{aligned}$$

and

$$\begin{aligned} E_{ij} &= E_i E_j, \\ g_{ij} &= E_i g_j + g_i, \\ L_{ij} &= E_i L_j E_i^\top + L_i. \end{aligned}$$

V. Numerical Experiment

In order to illustrate the benefit of parallelization we consider a simple tracking model (see, e.g., [5], [6]) with the state $x = \begin{pmatrix} u & v & \dot{u} & \dot{v} \end{pmatrix}^\top$, where (u, v) is the 2D position and $(\dot{u}, \dot{v})$ is the 2D velocity of the tracked object. From noisy measurements of the

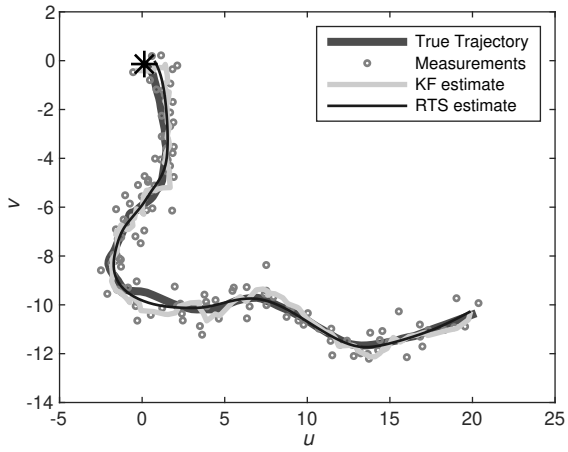

Fig. 2. Simulated trajectory from the linear tracking model in Eqs. (15) and (16) along with the Kalman filter (KF) and RTS smoother results.

position (u, v), we aim to solve the smoothing problem in order to determine the whole trajectory of the target.

The model has the form

$$\begin{aligned} x_k &= F\, x_{k-1} + q_{k-1}, \\ y_k &= H\, x_k + r_k, \end{aligned} \tag{15}$$

where $q_k \sim \mathrm{N}(0, Q)$, $r_k \sim \mathrm{N}(0, R)$, and

$$F = \begin{pmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \quad Q = q \begin{pmatrix} \frac{\Delta t^3}{3} & 0 & \frac{\Delta t^2}{2} & 0 \\ 0 & \frac{\Delta t^3}{3} & 0 & \frac{\Delta t^2}{2} \\ \frac{\Delta t^2}{2} & 0 & \Delta t & 0 \\ 0 & \frac{\Delta t^2}{2} & 0 & \Delta t \end{pmatrix}, \tag{16}$$

along with

$$H = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}, \qquad R = \begin{pmatrix} \sigma^2 & 0 \\ 0 & \sigma^2 \end{pmatrix}. \tag{17}$$

In our simulations we used the parameters $\sigma = 0.5$, $\Delta t = 0.1$, $q = 1$, and started the trajectory from a random Gaussian initial condition with mean $m_0 = \begin{pmatrix} 0 & 0 & 1 & -1 \end{pmatrix}^\top$ and covariance $P_0 = I_4$.

Fig. 2 shows a typical trajectory and measurements from the model defined by Eqs. (15) and (16) along with the Kalman filter and RTS smoother solutions. As the parallel algorithms produce exactly the same filter and smoothing solutions as the classic sequential algorithms, this result also illustrates the typical result produced by the proposed algorithms.

We now aim to evaluate the required number of floating point operations (flops) for generating the smoothing solution for this model. In order to do that, we run the sequential filter and smoothing methods (KF and RTS) as well as the proposed parallel algorithms (PKF and PRTS) over simulated data sets of different sizes and evaluate their span and work flops. The span flops here refers to the minimum number of floating point steps when the parallelizable operations in the algorithm are done in parallel – this corresponds to the actual execution time required to do the computations in a parallel computer. The work flops refers to the total number of operations that the parallel computer needs to perform – it measures the total energy required for the computations or equivalently the time required by the algorithm in a single-core computer. As the classic sequential KF and RTS algorithms are not parallelizable, their span and work flops are equal. The flops have been computed by estimating how many flops each of the matrix operations takes (multiplication, summation, LU-factorization) and incrementing the flops counter after every operation in the code.

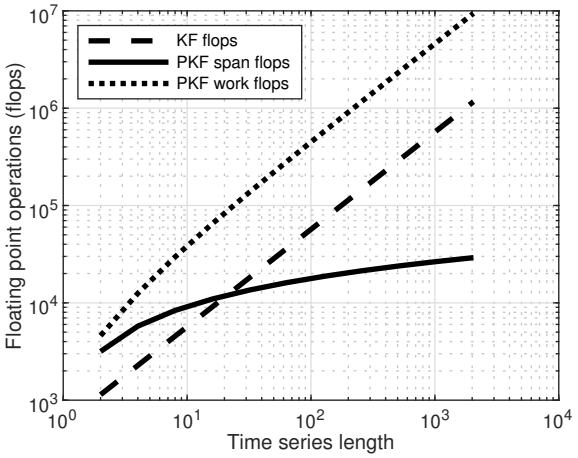

Fig. 3. The flops and the span and work flops for the sequential Kalman filter (KF) and the parallel Kalman filter (PKF).

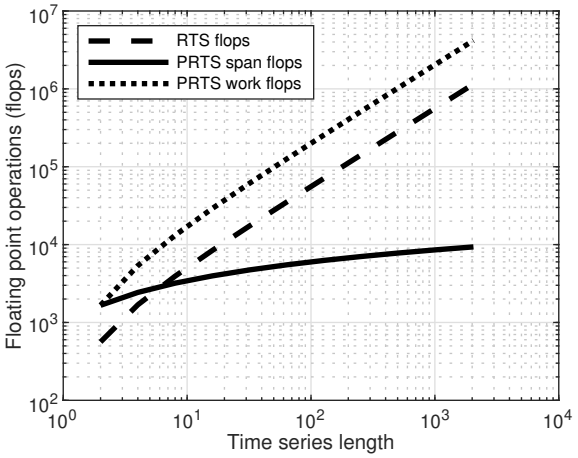

Fig. 4. The flops and the span and work flops for the (sequential) RTS smoother and the parallel RTS (PRTS) smoother.

Fig. 3 shows the flops required by the sequential KF along with the span flops and work flops required by the parallel Kalman filter algorithm. As expected, with small data set sizes the number of span flops required by the parallel KF is larger than that of the sequential KF, but already starting from time step count of around 20, the span flops is lower for the parallel KF. The logarithmic growth of the span flops in the parallel algorithm can be clearly seen while the number of flops for the sequential KF grows linearly. However, the work flops required by the parallel KF is approximately 8 times the flops of the sequential KF. This means that although the execution time for the parallel algorithms is smaller than for the sequential algorithms, they need to perform more floating point operations in total.

The flops required by the sequential RTS smoother along with the span flops and work flops required by the parallel RTS smoother are shown in Fig. 4. In this case, the parallel algorithm reaches the sequential algorithm speed already with data set of size less than 10. Furthermore, the total number of floating point operations required by the parallel algorithm is approximately 4 times the operations required by the sequential algorithm. The ratios of these total (work) operations for both the filter and smoother are shown in Fig. 5.

VI. Conclusion and Discussion

In this article we have proposed a novel general algorithmic framework for parallel computation of batch Bayesian filtering and

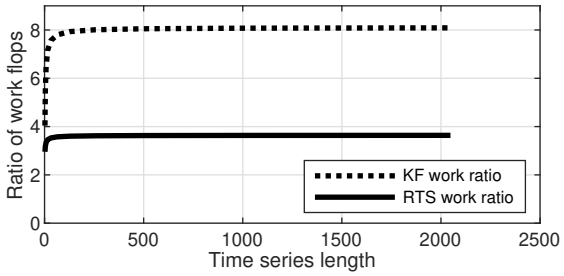

Fig. 5. Ratio of work flops for the parallel and sequential Kalman filter and the parallel and sequential RTS smoother.

smoothing solutions for state space models. The framework is based on formulating the computations in terms of associative operations between suitably defined elements such that the all-prefix-sums operation computed by a parallel-scan algorithm exactly produces the Bayesian filtering and smoothing solutions. The advantage of the framework is that the parallelization allows for performing the computations in $O(\log n)$ span complexity, where n is the number of data points, while sequential filtering and smoothing algorithms have an $O(n)$ complexity. Parallel versions of Kalman filters and Rauch–Tung–Striebel smoothers were derived as special cases of the framework. The computational advantages of the framework were illustrated in a numerical simulation.

A disadvantage of the proposed methodology is that although the wall-clock time of execution is significantly reduced, the total amount of computations (and hence required energy) is larger than with conventional sequential algorithms. Although the total amount of computations is only increased by a constant factor, in some systems, such as small-scale mobile systems, even if parallelization would be possible, it can be beneficial to use the classic algorithms. However, the speedup gain of the proposed approach is beneficial in applications such as data-assimilation based weather forecasting [24] and other spatio-temporal systems appearing, for example, in tomographic reconstruction [8] or machine learning [25], where the computations take a significant amount of time. In these systems, it is possible to dedicate the required amount of extra computational resources to gain the significant speedup provided by parallelization.

Although we have restricted our consideration to specific types parallel-scan algorithms, it is also possible to use other kinds of algorithms for computing the prefix sums corresponding to the Bayesian filtering and smoothing solutions. We could also select algorithms for given computer or network architectures, for minimizing the communication between the nodes, or for minimizing the energy consumption [26], [27]. The present formulation of the computations in terms of local associative operations is likely to have other applications beyond parallelization. For example, in decentralized systems, it is advantageous to be able to first perform operations locally and then combine them to produce the full state-estimation solution.

The proposed framework is also valid for discrete state spaces as well as for other state spaces provided that we consider the elements with the appropriate domain and replace the Lebesgue integrals by integrals with respect to the corresponding reference measure, e.g., counting measure in the case of discrete-state models.

The framework could be extended to non-linear and non-Gaussian models by replacing the exact Kalman filters and smoothers with iterated extended Kalman filters and smoothers [28], [29] or their sigma-point/numerical-integration versions such as posterior linearization filters and smoothers [30]–[32]. Possible future work also includes developing particle filter and smoother methods (see, e.g., [6]) for the present framework along with various other Bayesian filter and smoother approximations proposed in literature.

Appendix I

In this appendix, we prove the required results for Bayesian filtering: the associative property of the operator in Definition 2 and Theorem 3.

A. Associative property

In order to prove the associative property of $\otimes$ for filtering, we need to prove that for three elements (f_i, g_i), (f_j, g_j), $(f_k, g_k) \in \mathcal{F}$, the following relation holds

$$\begin{aligned}&\left[(f_i, g_i) \otimes (f_j, g_j)\right] \otimes (f_k, g_k) \\ &= (f_i, g_i) \otimes \left[(f_j, g_j) \otimes (f_k, g_k)\right]. \end{aligned} \tag{18}$$

We proceed to perform the calculations on both sides of the equation to check that they yield the same result.

1) Left-hand side: We use Definition 2 in the left-hand side of (18) and obtain

$$(f_{ij}, g_{ij}) \otimes (f_k, g_k) = (f_{ijk}, g_{ijk}),$$

where

$$\begin{aligned}&f_{ijk}(x \mid z) \\ &= \frac{\iint g_k(y) f_k(x \mid y) g_j(y') f_j(y \mid y') f_i(y' \mid z) \,\mathrm{d}y'\mathrm{d}y}{\iint g_k(y) g_j(y') f_j(y \mid y') f_i(y' \mid z) \,\mathrm{d}y'\mathrm{d}y}, \end{aligned} \tag{19}$$

and

$$\begin{aligned}&g_{ijk}(z) \\ &= g_{ij}(z) \int g_k(y) f_{ij}(y \mid z) \,\mathrm{d}y \\ &= g_i(z) \left[\int g_j(y) f_i(y \mid z) \,\mathrm{d}y\right] \\ &\quad \times \left[\int g_k(y) \left[\frac{\int g_j(y') f_j(y \mid y') f_i(y' \mid z) \,\mathrm{d}y'}{\int g_j(y') f_i(y' \mid z) \,\mathrm{d}y'}\right] \mathrm{d}y\right] \\ &= g_i(z) \iint g_k(y) g_j(y') f_j(y \mid y') f_i(y' \mid z) \,\mathrm{d}y'\mathrm{d}y. \end{aligned} \tag{20}$$

2) Right-hand side: We first use operator $\otimes$ to the elements with indices j and k in the right-hand side of (18), see Definition 2,

$$(f_j, g_j) \otimes (f_k, g_k) = (f_{jk}, g_{jk}),$$

where

$$\begin{aligned} f_{jk}(x \mid z) &= \frac{\int g_k(y) f_k(x \mid y) f_j(y \mid z) \,\mathrm{d}y}{\int g_k(y) f_j(y \mid z) \,\mathrm{d}y}, \\ g_{jk}(z) &= g_j(z) \int g_k(y) f_j(y \mid z) \,\mathrm{d}y. \end{aligned}$$

Then, the right-hand side of (18) becomes

$$(f_i, g_i) \otimes (f_{jk}, g_{jk}) = (f'_{ijk}, g'_{ijk}),$$

where

$$\begin{aligned}&f'_{ijk}(x \mid z) \\ &= \frac{\int g_{jk}(y) f_{jk}(x \mid y) f_i(y \mid z) \,\mathrm{d}y}{\int g_{jk}(y) f_i(y \mid z) \,\mathrm{d}y} \\ &= \frac{\int g_j(y) \left[\int g_k(y') f_k(x \mid y') f_j(y' \mid y) \,\mathrm{d}y'\right] f_i(y \mid z) \,\mathrm{d}y}{\int \left[g_j(y) \int g_k(y') f_j(y' \mid y) \,\mathrm{d}y'\right] f_i(y \mid z) \,\mathrm{d}y} \\ &= \frac{\iint g_j(y) g_k(y') f_k(x \mid y') f_j(y' \mid y) f_i(y \mid z) \,\mathrm{d}y'\mathrm{d}y}{\iint g_j(y) g_k(y') f_j(y' \mid y) f_i(y \mid z) \,\mathrm{d}y'\mathrm{d}y}, \end{aligned} \tag{21}$$

and

$$\begin{aligned} & g'_{ijk}(z) \\ &= g_i(z) \int g_{jk}(y)\, f_i(y \mid z)\, \mathrm{d}y \\ &= g_i(z) \int \left[g_j(y) \int g_k(y')\, f_j(y' \mid y)\, \mathrm{d}y' \right] f_i(y \mid z)\, \mathrm{d}y \\ &= g_i(z) \iint g_j(y)\, g_k(y')\, f_j(y' \mid y)\, f_i(y \mid z)\, \mathrm{d}y' \mathrm{d}y. \end{aligned} \tag{22}$$

In (19), (20), it is met that $f'_{ijk}(x \mid z) = f_{ijk}(x \mid z)$ and $g'_{ijk}(z) = g_{ijk}(z)$, which proves the associative property of $\otimes$ in Definition 2.

B. Proof of Theorem 3

In this appendix, we prove Theorem 3. We first prove by induction that

$$\begin{aligned} & a_{k-l} \otimes \cdots \otimes a_{k-1} \otimes a_k \\ &= \left(p(x_k \mid y_{k-l:k}, x_{k-l-1}), p(y_{k-l:k} \mid x_{k-l-1}) \right), \end{aligned} \tag{23}$$

for $l < k+1$. Relation (23) holds for $l = 0$ by definition of a_k. Then, assuming that

$$\begin{aligned} & a_{k-l+1} \otimes \cdots \otimes a_{k-1} \otimes a_k \\ &= \left(p(x_k \mid y_{k-l+1:k}, x_{k-l}), p(y_{k-l+1:k} \mid x_{k-l}) \right) \end{aligned} \tag{24}$$

holds, we need to prove that (23) holds.

We calculate the first element of $a_{k-l} \otimes b_{k-l+1}$, denoted by f_{ab}, where $b_{k-l+1} = a_{k-l+1} \otimes \cdots \otimes a_{k-1} \otimes a_k$. We have

$$\begin{aligned} & f_{ab}(x_k \mid x_{k-l}) \\ &= \frac{\int p(y_{k-l+1:k}, x_k \mid x_{k-l})\, p(x_{k-l} \mid y_{k-l}, x_{k-l-1})\, \mathrm{d}x_{k-l}}{p(y_{k-l+1:k} \mid y_{k-l}, x_{k-l-1})} \\ &= \frac{p(y_{k-l+1:k}, x_k \mid y_{k-l}, x_{k-l-1})}{p(y_{k-l+1:k} \mid y_{k-l}, x_{k-l-1})} \\ &= p(x_k \mid y_{k-l:k}, x_{k-l-1}). \end{aligned}$$

Function f_{ab} corresponds to the first element of (23), as required. We further get

$$\begin{aligned} g_{ab}(x_{k-l-1}) &= p(y_{k-l} \mid x_{k-l-1}) \int p(y_{k-l+1:k} \mid x_{k-l}) \\ &\quad \times p(x_{k-l} \mid y_{k-l}, x_{k-l-1})\, \mathrm{d}x_{k-l} \\ &= p(y_{k-l} \mid x_{k-l-1})\, p(y_{k-l+1:k} \mid y_{k-l}, x_{k-l-1}) \\ &= p(y_{k-l:k} \mid x_{k-l-1}). \end{aligned}$$

Function g_{ab} corresponds to the second element of (23), as required.

Substituting $l = k+2$ into (23), we obtain

$$\begin{aligned} & a_2 \otimes \cdots \otimes a_k \\ &= \left(p(x_k \mid y_{2:k}, x_1), p(y_{2:k} \mid x_1) \right). \end{aligned} \tag{25}$$

We now calculate the first element of $a_1 \otimes [a_2 \otimes \cdots \otimes a_k]$, denoted as f_{1k}, where a_1 is given in Theorem 3:

$$\begin{aligned} & f_{1k}(x_k \mid x_0) \\ &= \frac{\int p(y_{2:k} \mid x_1)\, p(x_k \mid y_{2:k}, x_1)\, p(x_1 \mid y_1)\, \mathrm{d}x_1}{\int p(y_{2:k} \mid x_1)\, p(x_1 \mid y_1)\, \mathrm{d}x_1} \\ &= \frac{\int p(y_{2:k}, x_k \mid x_1, y_1)\, p(x_1 \mid y_1)\, \mathrm{d}x_1}{p(y_{2:k} \mid y_1)} \\ &= \frac{p(y_{2:k}, x_k \mid y_1)}{p(y_{2:k} \mid y_1)} \\ &= p(x_k \mid y_{1:k}). \end{aligned} \tag{26}$$

The second element of $a_1 \otimes [a_2 \otimes \cdots \otimes a_k]$, denoted as g_{1k} is

$$\begin{aligned} & g_{1k}(x_0) \\ &= p(y_1) \int p(y_{2:k} \mid x_1)\, p(x_1 \mid y_1)\, \mathrm{d}x_1 \\ &= p(y_{1:k}). \end{aligned} \tag{27}$$

Results (26) and (27) finish the proof of Theorem 3.

APPENDIX II

In this appendix, we prove the required results for Bayesian smoothing: the associative property of the operator in Definition 5 and Theorem 6.

A. Associative property

In order to prove the associative property of $\otimes$ for filtering, we need to prove that, for three elements a_i, a_j, $a_k \in \mathcal{S}$, the following relation holds:

$$\left[a_i \otimes a_j \right] \otimes a_k = a_i \otimes \left[a_j \otimes a_k \right]. \tag{28}$$

We proceed to perform the calculations on both sides of the equation to check that they yield the same result.

1) Left-hand side: We apply the operator in Definition 5 on the left-hand side of (28) to obtain

$$a_{ij} \otimes a_k = a_{ijk},$$

where

$$\begin{aligned} a_{ijk}(x \mid z) &= \int a_{ij}(x \mid y)\, a_k(y \mid z)\, \mathrm{d}y \\ &= \iint a_i(x \mid y')\, a_j(y' \mid y)\, a_k(y \mid z)\, \mathrm{d}y \mathrm{d}y'. \end{aligned} \tag{29}$$

2) Right-hand side: We first calculate $a_j \otimes a_k$ using the operator in Definition 5 which gives

$$a_{jk}(x \mid z) = \int a_j(x \mid y)\, a_k(y \mid z)\, \mathrm{d}y.$$

Then, we calculate the right hand side of (28) we have that

$$a_i \otimes a_{jk} = a'_{ijk},$$

where

$$\begin{aligned} a'_{ijk}(x \mid z) &= \int a_i(x \mid y)\, a_{jk}(y \mid z)\, \mathrm{d}y \\ &= \iint a_i(x \mid y)\, a_j(y \mid y')\, a_k(y' \mid z)\, \mathrm{d}y' \mathrm{d}y. \end{aligned} \tag{30}$$

We can see that a_{ijk} in (29) is equal to a'_{ijk} in (30), which proves the associative property of the operator in Definition 5.

B. Proof of Theorem 6

In this appendix, we prove Theorem 6. We first prove by induction that

$$\begin{aligned} & a_k \otimes \cdots \otimes a_{k+l} \\ &= p(x_k \mid y_{1:k+l}, x_{k+l+1}), \end{aligned} \tag{31}$$

for $l < n-k$. Relation (31) holds for $l = 0$ by definition of a_k. Then, assuming that

$$\begin{aligned} & a_k \otimes \cdots \otimes a_{k+l-1} \\ &= p(x_k \mid y_{1:k+l-1}, x_{k+l}) \end{aligned} \tag{32}$$

holds, we need to prove that (23) holds.

We use a_{k+l} in Theorem 6 to calculate

$$\begin{aligned}
&[a_k \otimes \cdots \otimes a_{k+l-1}] \otimes a_{k+l} \\
&= \int p\left(x_k \mid y_{1:k+l-1}, x_{k+l}\right) p\left(x_{k+l} \mid y_{1:k+l}, x_{k+l+1}\right) \mathrm{d}x_{k+l} \\
&= \int p\left(x_k \mid y_{1:k+l}, x_{k+l}, x_{k+l+1}\right) \\
&\quad \times p\left(x_{k+l} \mid y_{1:k+l}, x_{k+l+1}\right) \mathrm{d}x_{k+l} \\
&= \int p\left(x_k, x_{k+l} \mid y_{1:k+l}, x_{k+l+1}\right) \mathrm{d}x_{k+l} \\
&= p\left(x_k \mid y_{1:k+l}, x_{k+l+1}\right).
\end{aligned}$$

This proves (31).

If $l = n - k - 1$ and a_n as in Theorem 6, we have

$$\begin{aligned}
&[a_k \otimes \cdots \otimes a_{n-1}] \otimes a_n \\
&= \int p\left(x_k \mid y_{1:n-1}, x_n\right) p\left(x_n \mid y_{1:n}\right) \mathrm{d}x_n \\
&= \int p\left(x_k \mid y_{1:n}, x_n\right) p\left(x_n \mid y_{1:n}\right) \mathrm{d}x_n \\
&= p\left(x_k \mid y_{1:n}\right).
\end{aligned}$$

This result finishes the proof of Theorem 6.

Appendix III

In this appendix, we prove Lemma 8. We have the following easily verifiable identities:

$$\begin{aligned}
&\mathrm{N}_I(y; \eta, J)\mathrm{N}(y; m, C) \\
&\propto \mathrm{N}(y; [J + C^{-1}]^{-1}[\eta + C^{-1}m], [J + C^{-1}]^{-1})
\end{aligned}$$

and

$$\mathrm{N}_I(y; \eta, J)\mathrm{N}_I(y; \eta', J') \propto \mathrm{N}_I(y; \eta + \eta', J + J').$$

We also have

$$\begin{aligned}
&\int \mathrm{N}_I(y; \eta, J)\mathrm{N}(y; Az + b, C)\mathrm{d}y \\
&\propto \mathrm{N}_I(z; A^\top [I + JC]^{-1}(\eta - Jb), A^\top [I + JC]^{-1} JA).
\end{aligned}$$

By using Definition 2 for f_{ij} and g_{ij} together with parameterizations in Lemma 8, elementary computations lead to (13) and (14).

Acknowledgment

The authors would like to thank Academy of Finland for financial support.

References

[1] T. Rauber and G. Rünger, *Parallel programming: For multicore and cluster systems*, 2nd ed. Springer, 2013.
[2] S. Cook, *CUDA programming: a developer's guide to parallel computing with GPUs*. Morgan Kaufmann, 2013.
[3] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*, 3rd ed. MIT Press, 2009.
[4] A. H. Jazwinski, *Stochastic Processes and Filtering Theory*. Academic Press, New York, 1970.
[5] Y. Bar-Shalom, X.-R. Li, and T. Kirubarajan, *Estimation with Applications to Tracking and Navigation*. Wiley, New York, 2001.
[6] S. Särkkä, *Bayesian Filtering and Smoothing*. Cambridge University Press, 2013.
[7] O. Cappé, E. Moulines, and T. Rydén, *Inference in Hidden Markov Models*, ser. Springer Series in Statistics. New York, NY: Springer-Verlag, 2005.
[8] J. Kaipio and E. Somersalo, *Statistical and Computational Inverse Problems*. Springer, 2005.
[9] S. Särkkä, M. A. Álvarez, and N. D. Lawrence, "Gaussian process latent force models for learning and stochastic control of physical systems," *IEEE Transactions on Automatic Control*, 2019.
[10] Y. C. Ho and R. C. K. Lee, "A Bayesian approach to problems in stochastic estimation and control," *IEEE Transactions on Automatic Control*, vol. 9, no. 4, pp. 333–339, 1964.
[11] T. D. Barfoot, C. H. Tong, and S. Särkkä, "Batch continuous-time trajectory estimation as exactly sparse Gaussian process regression," in *Proceedings of Robotics: Science and Systems (RSS)*, 2014.
[12] A. Grigorievskiy, N. Lawrence, and S. Särkkä, "Parallelizable sparse inverse formulation Gaussian processes (SpInGP)," in *Proceedings of IEEE International Workshop on Machine Learning for Signal Processing (MLSP)*, 2017.
[13] P. M. Lyster, S. E. Cohn, R. Ménard, L. P. Chang, S. J. Lin, and R. G. Olsen, "Parallel implementation of a Kalman filter for constituent data assimilation," *Monthly Weather Review*, vol. 125, no. 7, pp. 1674–1686, 1997.
[14] G. Evensen, "The ensemble Kalman filter: Theoretical formulation and practical implementation," *Ocean dynamics*, vol. 53, no. 4, pp. 343–367, 2003.
[15] A. Lee, C. Yau, M. B. Giles, A. Doucet, and C. C. Holmes, "On the utility of graphics cards to perform massively parallel simulation of advanced Monte Carlo methods," *Journal of Computational and Graphical Statistics*, vol. 19, no. 4, pp. 769–789, 2010.
[16] O. Rosen and A. Medvedev, "Efficient parallel implementation of state estimation algorithms on multicore platforms," *IEEE Transactions on Control Systems Technology*, vol. 21, no. 1, pp. 107–120, 2013.
[17] M. E. Liggins, C.-Y. Chong, I. Kadar, M. G. Alford, V. Vannicola, and S. Thomopoulos, "Distributed fusion architectures and algorithms for target tracking," *Proceedings of the IEEE*, vol. 85, no. 1, pp. 95–107, 1997.
[18] R. E. Ladner and M. J. Fischer, "Parallel prefix computation," *Journal of the ACM*, vol. 27, no. 4, pp. 831–838, 1980.
[19] G. E. Blelloch, "Scans as primitive parallel operations," *IEEE Transactions on Computers*, vol. 38, no. 11, pp. 1526–1538, 1989.
[20] ——, "Prefix sums and their applications," School of Computer Science, Carnegie Mellon University, Tech. Rep. CMU-CS-90-190, 1990.
[21] R. E. Kalman, "A new approach to linear filtering and prediction problems," *Transactions of the ASME, Journal of Basic Engineering*, vol. 82, no. 1, pp. 35–45, 1960.
[22] H. E. Rauch, F. Tung, and C. T. Striebel, "Maximum likelihood estimates of linear dynamic systems," *AIAA Journal*, vol. 3, no. 8, pp. 1445–1450, 1965.
[23] B. D. O. Anderson and J. B. Moore, *Optimal Filtering*. Prentice-Hall, 1979.
[24] N. Cressie and C. K. Wikle, *Statistics for Spatio-Temporal Data*. John Wiley & Sons, 2011.
[25] S. Särkkä, A. Solin, and J. Hartikainen, "Spatiotemporal learning via infinite-dimensional Bayesian filtering and smoothing," *IEEE Signal Processing Magazine*, vol. 30, no. 4, pp. 51–61, 2013.
[26] A. Grama, V. Kumar, A. Gupta, and G. Karypis, *Introduction to parallel computing*, 2nd ed. Pearson Education, 2003.
[27] P. Sanders and J. L. Träff, "Parallel prefix (scan) algorithms for MPI," in *Recent Advances in Parallel Virtual Machine and Message Passing Interface. EuroPVM/MPI 2006. Lecture Notes in Computer Science*, B. Mohr, J. Träff, J. Worringen, and J. Dongarra, Eds. Springer, 2006, vol. 4192.
[28] B. M. Bell and F. W. Cathey, "The iterated Kalman filter update as a Gauss–Newton method," *IEEE Transactions on Automatic Control*, vol. 38, no. 2, pp. 294–297, 1993.
[29] B. M. Bell, "The iterated Kalman smoother as a Gauss–Newton method," *SIAM Journal on Optimization*, vol. 4, no. 3, pp. 626–636, 1994.
[30] A. F. García-Fernández, L. Svensson, M. R. Morelande, and S. Särkkä, "Posterior linearisation filter: principles and implementation using sigma points," *IEEE Transactions on Signal Processing*, vol. 63, no. 20, pp. 5561–5573, 2015.
[31] A. F. García-Fernández, L. Svensson, and S. Särkkä, "Iterated posterior linearization smoother," *IEEE Transactions on Automatic Control*, vol. 62, no. 4, pp. 2056–2063, 2017.
[32] F. Tronarp, A. F. García-Fernández, and S. Särkkä, "Iterative filtering and smoothing in non-linear and non-Gaussian systems using conditional moments," *IEEE Signal Processing Letters*, vol. 25, no. 3, pp. 408–412, 2018.